"EXPRESS MAIL" Mailing Label No.
EL840960590 US
Date of Deposit: March 30, 2001

MANAGEMENT OF CO-PROCESSOR INFORMATION BY INTEGRATING NON-PROGRAM INFORMATION WITH PROGRAM INFORMATION

This application claims the priority under 35 U.S.C. 119(e)(1) of copending U.S. provisional application number 60/194,258 filed on April 3, 2000, incorporated herein by reference.

FIELD OF THE INVENTION

The invention relates generally to data processing and, more particularly, to management of co-processor information.

BACKGROUND OF THE INVENTION

Data processing systems (including single-chip systems) which utilize one or more host processors (for example microprocessors) and one or more co-processors (for example further microprocessors or digital signal processors) under control of the host processor(s) are well known in the art. In such systems, a co-processor is typically configured to perform desired functions by storing associated program information in a ROM (or other program memory) of the co-processor. The operational capability of the

co-processor thus corresponds to, and is limited by, the functionality defined by the program(s) stored in its ROM.

It therefore becomes advantageous to provide systems wherein the host processor(s) can dynamically download desired programs to the co-processor(s) in response to user requests. For example, a user of a mobile telephone might initially wish to configure the mobile telephone's co-processor(s) to perform speech coding functions in support of voice communications, and might later wish to configure the co-processor(s) to perform functions in support of wireless internet access. Accordingly, in this example, the host processor(s) would first download into the co-processor(s) a program or programs for configuring the co-processor(s) as a speech coder, and would thereafter download a program or programs for configuring the co-processor(s) as an internet access port.

Different programs (also referred to herein as objects) have different platform requirements, for example, program memory size, data memory size, clock speed, etc. Therefore, a given program may not be suitable for execution on a given co-processor. The suitability of a given program for execution on a given co-processor can be determined by a host processor before downloading the program to the co-processor, provided the program developer can provide to the host processor information indicative of the platform requirements of the program. In systems with multiple co-processors, the host processor may be able to use the platform requirement information at runtime to identify an available co-processor which satisfies the platform requirements of the

program. The host processor can then download the program to the identified co-processor.

In conventional practice, non-program information has typically been stored separately from the executable file that contains the program. This practice disadvantageously dictates the use of an auxiliary data source for non-program information, for example an SQL engine, Microsoft's registry, or even a simple text file. This also disadvantageously requires that two separate files be handled, the executable and the non-program information file.

It is desirable in view of the foregoing to provide for efficient communication of non-program information, such as platform requirement information, from the program developer to the host processor at runtime.

The present invention provides for efficient communication of non-program information, e.g. platform requirement information, from the program developer to the host processor at runtime. The non-program information is integrated along with the corresponding program information into an executable file which is used by the host processor to download the program information to a selected co-processor.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 diagrammatically illustrates pertinent portions of exemplary embodiments of a data processing system according to the invention.

FIGURE 2 diagrammatically illustrates an exemplary procedure for producing a virtual database within the file storage facility of FIGURE 1.

FIGURE 3 illustrates portions of the process of FIGURE 2 in greater detail.

FIGURE 4 illustrates in tabular format examples of non-program information which can be provided by a developer and stored in the virtual database of FIGURE 2.

FIGURE 5 diagrammatically illustrates pertinent portions of exemplary embodiments of the application programming interface of FIGURES 1 and 2.

FIGURE 6 illustrates exemplary operations according to the invention.

FIGURE 7 illustrates an example of an executable file which can be produced by the process of FIGURES 2 and 3 and stored in the file storage facility of FIGURES 1, 3 and 5.

FIGURE 8 diagrammatically illustrates, by comparison with FIGURE 1, exemplary advantages of the present invention.

DETAILED DESCRIPTION

FIGURE 1 diagrammatically illustrates pertinent portions of exemplary embodiments of a data processing system according to the invention. Examples of the data processing system include wireless telephones, laptop computers, and set-top boxes.

The exemplary system of FIGURE 1 includes a host processor 11 (for example a microprocessor) and one or more co-processors 13 (for example additional microprocessors and/or DSPs). The processors 11 and 13 can be embedded together in a single integrated circuit chip, or can be provided on separate integrated circuit chips. A man-machine interface (MMI) 12, for example a keyboard/keypad, visual display, etc. permits a user to access user applications 14 associated with the host processor 11. When a user application determines that a co-processor should execute a particular function, the application directs a server 15 in the host processor 11 to obtain program information to be downloaded from the server 15 to the co-processor, and then used by the co-processor in performing the desired function. In response to the request from the user application 14, the server 15 uses an application programming interface (API) 16 to retrieve the program information from a file storage facility (e.g. a file system or other file storage mechanism) 17 where executable files are stored.

According to the invention, a given executable file stored in the file storage facility 17 includes not only program information which the co-processor uses to perform the desired function, but also includes non-program information associated with the program information. For example, the non-program information could include platform

requirement information such as described above, setup parameters, or other general properties of the program. The API 16 distinguishes the program information from the non-program information, and provides both sets of information to the server 15. Based on the non-program information, the server can, for example, make a determination as to which of a plurality of available co-processors is suitable for execution of the desired program, and can then forward the program information to the selected co-processor.

FIGURE 2 diagrammatically illustrates exemplary manners in which the aforementioned non-program information can be configured. As shown in FIGURE 2, during a program development phase, the developer uses software tools to configure a virtual database 17' which stores object (i.e. program) attributes such as, for example, platform requirement information, setup parameters, etc. The virtual database 17' is provided within the storage facility 17 of FIGURE 1 as will be described in more detail below. A configuration tool 21 provides the attribute (non-program) information. A UUID generation tool 23 provides to the configuration tool 21 a universally unique identifier (UUID) for identifying each set of attribute information stored in the virtual database 17'. The UUIDs are included in the attribute information provided by the configuration tool 21. After the virtual database 17' has been established during the development phase, the API 16 accesses the stored object attributes and provides them to the server 15 during the runtime phase.

FIGURE 3 diagrammatically illustrates an exemplary process for providing the virtual database 17' of FIGURE 2 in the storage facility 17 of FIGURE 1. In the example

of FIGURE 3, the configuration tool 21 is based on Texas Instruments Incorporated's commercially available Graphical Configuration Tool (GCONF). This tool is a Windows GUI application that presents different configurable data modules in a manner similar to Windows Explorer. The developer can conduct a dialogue in conventional fashion with the GCONF tool, inputting the desired attributes for each program (or object) in FIGURE 3. The programs are designated as obj1, obj2, ...objn in FIGURE 3. Using conventional techniques, the GCONF tool can be suitably programmed to convert the input attribute information into information which is suitable for integration into an executable file, for example a COFF (Common Object File Format) executable file. After the attribute information has been input into a suitable configuration file in the GCONF tool 21 during the aforementioned dialogue, the developer uses the GCONF "file/save" command, which prompts the GCONF tool to automatically generate header files, assembly macros and linker command files based on the attribute information provided by the developer during the dialogue process. The linker command files will contain the attribute information in a format suitable for integration into an executable file.

Each of the assembly/linker files 32, 33 and 34 (which include the aforementioned linker command files) is combined with its associated program information (i.e., code and data) at 36, 37 and 38, which program information is contained in conventional executable files (e.g. COFF files). The combining operation can be performed by a conventional compiler/linker 31. The compiler/linker 31 combines the data in the assembly/linker files 32, 33 and 34 with the program

information from the files 36, 37 and 38, respectively, to produce corresponding executable files, in this example COFF executable files 39, 40 and 41, that include both program information (from 36, 37 and 38) and non-program information (from 32, 33 and 34).

Each of the executable files 39, 40 and 41 illustrated in FIGURE 3 includes program information and non-program information, as illustrated generally by the example of FIGURE 7. The executable file 40 illustrated in FIGURE 7 includes program information (code and data) and corresponding non-program information, for example platform requirement information for the program, as described above. Although the example executable file 40 of FIGURE 7 includes only a single program and its associated non-program information, other executable files in the storage facility 17 could include code and data corresponding to a plurality of programs, together with a plurality of sets of non-program information respectively corresponding to the plurality of programs. The non-program information included in the various executable files 39, 40 and 41 in FIGURE 3 constitutes the virtual database 17' of FIGURE 2, more particularly a virtual database including non-program information corresponding to the various programs stored in the storage facility 17.

FIGURE 4 shows examples of attribute information associated with an exemplary codec node (i.e., codec program). As shown, the aforementioned UUID can be obtained by the developer (from the tool 23 of FIGURE 2) and provided as attribute information for the codec program.

Referring again to FIGURE 1, when the server 15 begins the process of loading an executable file onto a coprocessor for the first time, the API 16 will record the file path of the executable file. In some embodiments, the API 16 performs data retrieval through a parser 51 as illustrated in FIGURE 5. The parser 51 uses the UUID information described above to identify uniquely each program in the storage facility 17, and to identify data sections within the executable files wherein the corresponding non-program information is stored.

Executable files that conform to COFF, for example the COFF utilized by Texas Instruments Incorporated, support non-downloadable data regions. Using this feature of COFF, the compiler/linker 31 of FIGURE 3 automatically stores the non-program information within the non-downloadable data regions of the COFF executable files. Thus, the parser 51 will search through the COFF executable files within the storage facility 17, comparing the UUIDs of the non-downloadable data sections with the UUID provided to the parser 51 by the user (via the server 15). When the parser finds a non-downloadable data section UUID match, the non-program information from that section can, in some embodiments, be loaded into a corresponding data structure in the API 16. The non-program information can be provided to the server 15 along with the corresponding program information read from the storage facility 17, whereupon the server 15 can utilize conventional techniques to, for example, evaluate whether a given co-processor is suitable for execution of the desired program and/or to setup/configure the co-processor to execute the desired program. The parser 51 can be used to determine

the file path information described above, and this information can be stored in an OTC (Object to COFF) map 53. This map 53 can thereafter use the user-provided UUID information to map the various programs to their corresponding COFF files.

FIGURE 6 illustrates exemplary operations of the present invention. The program code and data is provided at 61, and the related non-program information is provided at 62. At 63, the non-program information is configured for inclusion in an executable file. At 64, the configured information is integrated into an executable file together with the program code and data. When it is desired at 65 to download the program from the host processor to a co-processor, the server at 66 obtains the executable file contents, and uses the non-program information to select the co-processor, after which the program can be downloaded into the co-processor at 67.

FIGURE 8 is provided to illustrate by comparison exemplary advantages of the invention described above with respect to FIGURES 1-7. FIGURE 8 diagrammatically illustrates the consequences of the lack of database standardization across different target operating systems. Whereas the invention described above with respect to FIGURES 1-7 is clearly cooperable with multiple target operating systems while using only a single API design and a single (virtual) database configuration, FIGURE 8 illustrates that, without the invention of FIGURES 1-7, multiple target operating systems could be supported only by multiple corresponding database access APIs, one database access API for each OS-specific database. As a result, the overall complexity of the system would increase significantly as clearly shown by a comparison of FIGURES 1 and 8.

It should also be noted that the invention described above with respect to FIGURES 1-7 provides a unique data access approach inasmuch as no other database server will be able to access the data in the above-described virtual database 17' unless, for example, that server has access to the UUIDs that are needed to access the data in the virtual database 17'.

The above-described integration of non-program information with program information in an executable file permits non-program information to be communicated from the developer to the server of the host processor in an efficient manner, and without increasing the size of the runtime program. This is accomplished by, for example, taking advantage of the non-downloadable data section feature of COFF executables. The invention eliminates the need for an auxiliary database on the host processor, thus saving the resources required by a traditional database, which is particularly advantageous for resource-constrained systems such as a system on a chip. The invention further simplifies the process of downloading a program to a co-processor because both program and non-program information can be provided in a single file, thereby advantageously avoiding the conventional requirement of handling two separate files. Also, as described above with respect to FIGURE 8, the invention provides compatibility across multiple platforms with far less complexity than would result through application of conventional techniques.

Although exemplary embodiments of the invention are described above in detail, this does not limit the scope of the invention, which can be practiced in a variety of embodiments.